

# dabl

Automatic ML with a human in the loop

## Andreas Müller

Associate Research Scientist  
Columbia University  
Scikit-learn Technical Committee

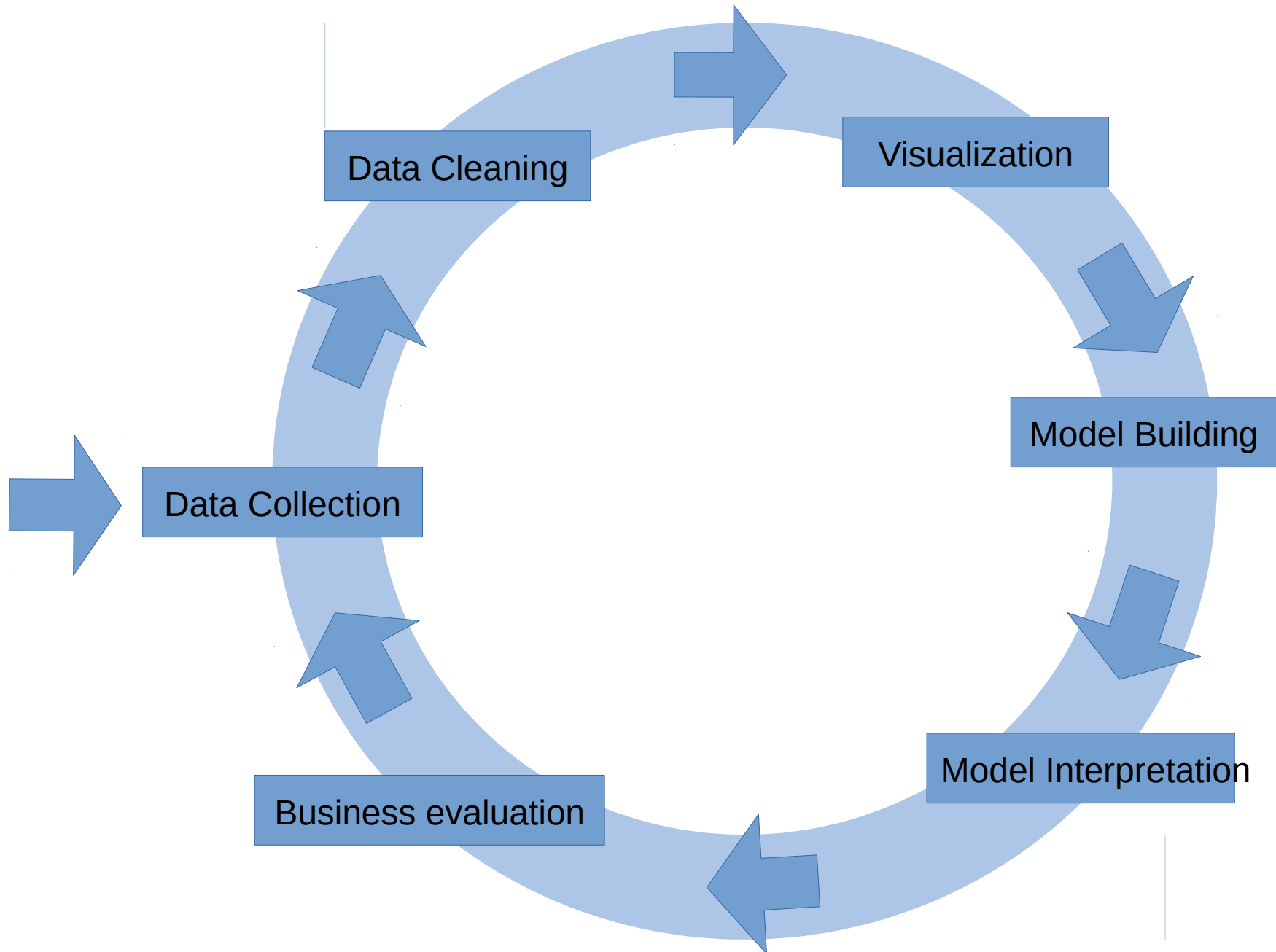


Alfred P. Sloan  
FOUNDATION



COLUMBIA UNIVERSITY  
IN THE CITY OF NEW YORK

# A real world ML workflow



# ML with sklearn & pandas

```
import pandas as pd
import seaborn as sns
```

```
data = pd.read_csv("adult.csv", index_col=0)
```

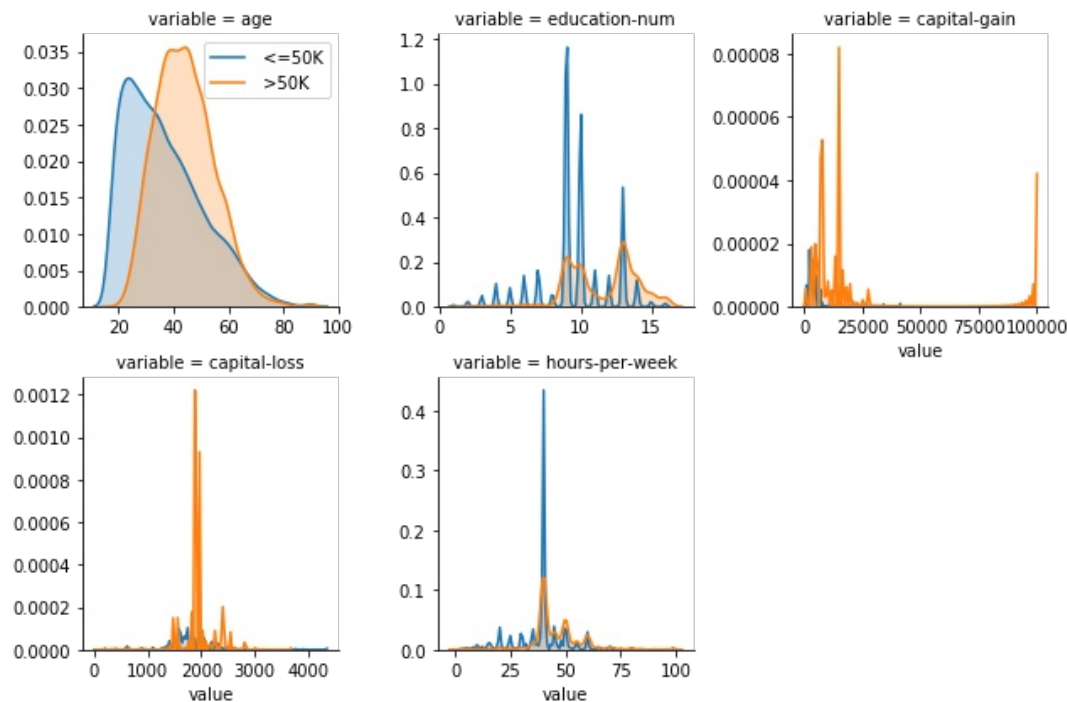
```
cols = data.columns[data.dtypes != object].tolist() + ['income']
```

```
df = data.loc[:, cols].melt("income")
```

```
g = sns.FacetGrid(df, col='variable', hue='income',
                  sharey=False, sharex=False, col_wrap=3)
```

```
g = g.map(sns.kdeplot, "value", shade=True)
```

```
g.axes[0].legend()
```



# ML with sklearn & pandas

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

categorical_columns = data_features.dtypes == object

cont_pipe = Pipeline([('scaler', StandardScaler()),
                      ('imputer', SimpleImputer(strategy='median', add_indicator=True))])
cat_pipe = Pipeline([('ohe', OneHotEncoder(handle_unknown='ignore')),
                     ('imputer', SimpleImputer(strategy='most_frequent', add_indicator=True))])

pre = ColumnTransformer([('categorical', cat_pipe, categorical_columns),
                        ('continuous', cont_pipe, ~categorical_columns),
                        ])

model = Pipeline([('preprocessing', pre), ('clf', LogisticRegression())])
param_grid = {'clf__C': np.logspace(-3, 3, 7)}
grid_search = GridSearchCV(model, param_grid=param_grid)
grid_search.fit(X_train, y_train)
```

---

# Current Automatic ML frameworks

## Example

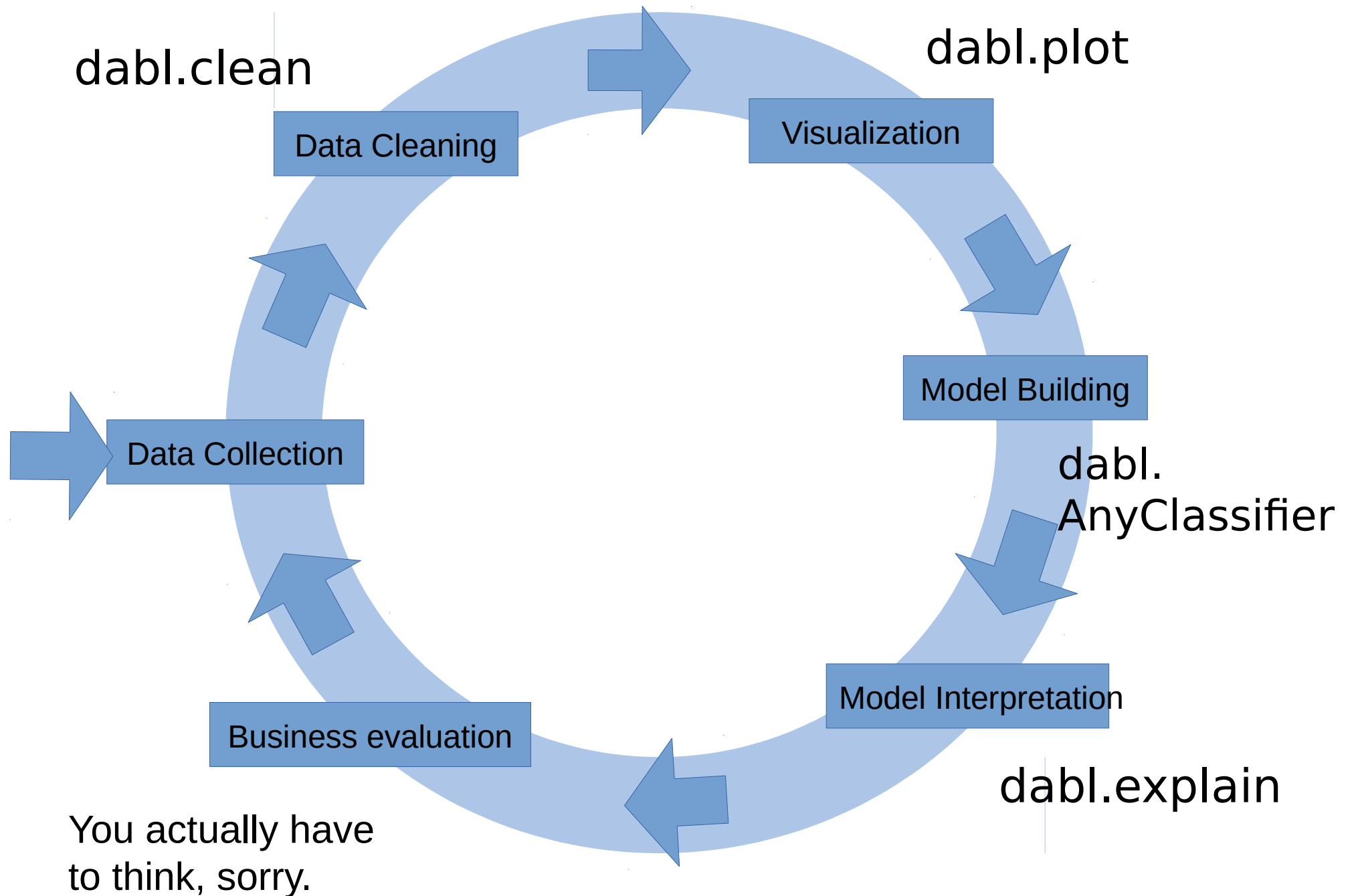
```
>>> import autosklearn.classification
>>> import sklearn.model_selection
>>> import sklearn.datasets
>>> import sklearn.metrics
>>> X, y = sklearn.datasets.load_digits(return_X_y=True)
>>> X_train, X_test, y_train, y_test = \
        sklearn.model_selection.train_test_split(X, y, random_state=1)
>>> automl = autosklearn.classification.AutoSklearnClassifier()
>>> automl.fit(X_train, y_train)
>>> y_hat = automl.predict(X_test)
>>> print("Accuracy score", sklearn.metrics.accuracy_score(y_test, y_hat))
```

This will run for one hour and should result in an accuracy above 0.98.

***A NEW HOPE***



data  
analysis  
baseline  
library  
***A NEW HOPE***





# Data cleaning & preprocessing

```
import dabl
ames_df = dabl.datasets.load_ames()
ames_df.head()
```

	Order	PID	MS SubClass	MS Zoning	Lot Frontage	Lot Area	Street	Alley	Lot Shape	Land Contour	...	Pool Area	Pool QC	Fence	Misc Feature	Misc Val	Mo Sold	Yr Sold	Sale Type	Sale Condition
0	1	526301100	20	RL	141.0	31770	Pave	NaN	IR1	Lvl	...	0	NaN	NaN	NaN	0	5	2010	WD	Normal
1	2	526350040	20	RH	80.0	11622	Pave	NaN	Reg	Lvl	...	0	NaN	MnPrv	NaN	0	6	2010	WD	Normal
2	3	526351010	20	RL	81.0	14267	Pave	NaN	IR1	Lvl	...	0	NaN	NaN	Gar2	12500	6	2010	WD	Normal
3	4	526353030	20	RL	93.0	11160	Pave	NaN	Reg	Lvl	...	0	NaN	NaN	NaN	0	4	2010	WD	Normal
4	5	527105010	60	RL	74.0	13830	Pave	NaN	IR1	Lvl	...	0	NaN	MnPrv	NaN	0	3	2010	WD	Normal

5 rows × 82 columns

```
clean_df = dabl.clean(ames_df, verbose=2)
```

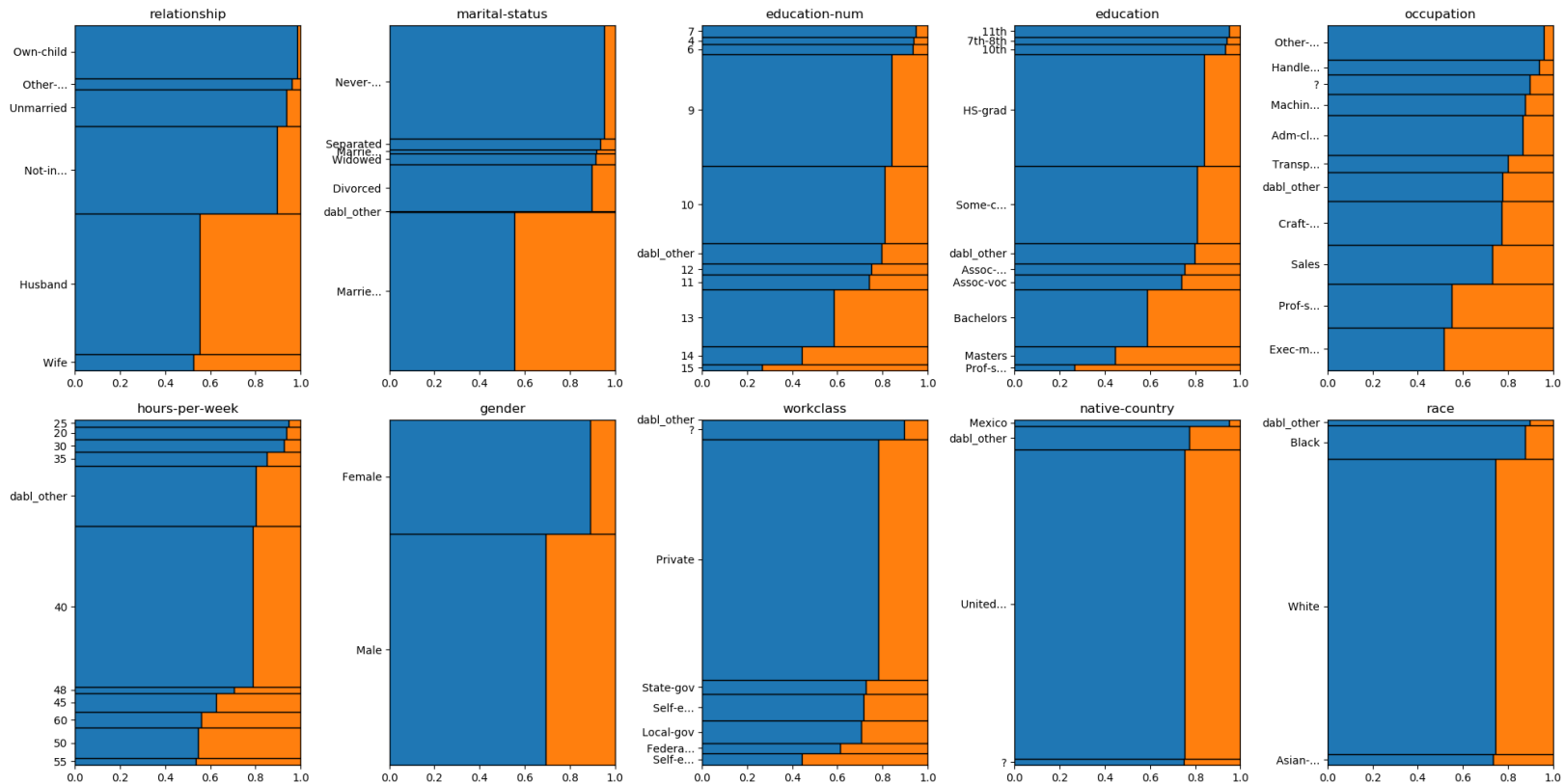
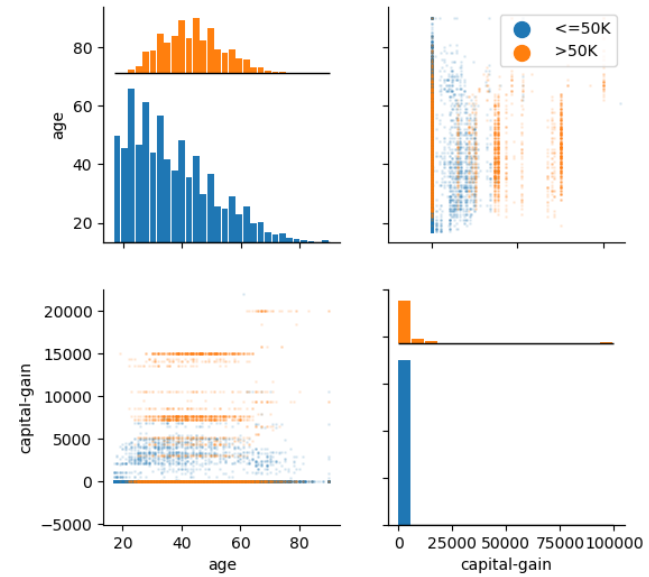
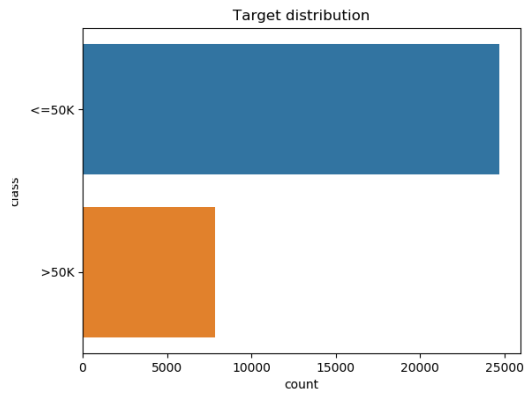
Detected feature types:  
11 float, 28 int, 43 object, 0 date, 0 other  
Interpreted as:  
continuous 23  
dirty\_float 0  
low\_card\_int 6  
categorical 40  
date 0  
free\_string 0  
useless 13  
dtype: int64  
WARN dropped useless columns: ['Order', 'Street', 'Utilities', 'Land Slope', 'Condition 2', 'Roof Matl', 'Heating', 'Low Qual Fin SF', 'Kitchen AbvGr', 'Garage Cond', '3Ssn Porch', 'Pool Area', 'Misc Val']

# dabl.clean

- Detect types (can overwrite)
- Detect Missing / rare values
- Detect ordinal vs categorical
- Detect near-constant
- Detect index

Visualization

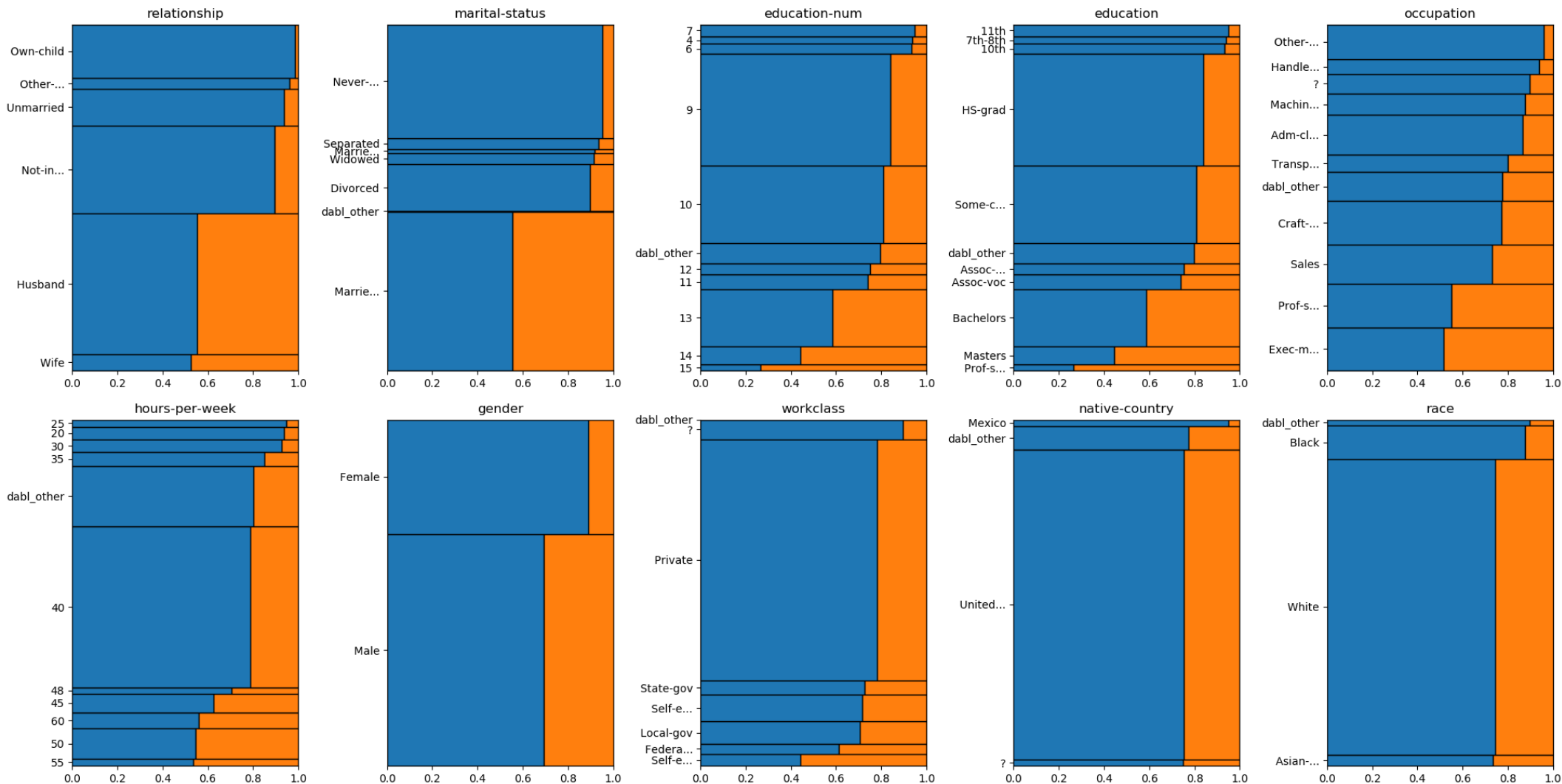
```
data = pd.read_csv("adult.csv", index_col=0)
plot(data, 'income', scatter_alpha=.1)
```



# Univariate plots

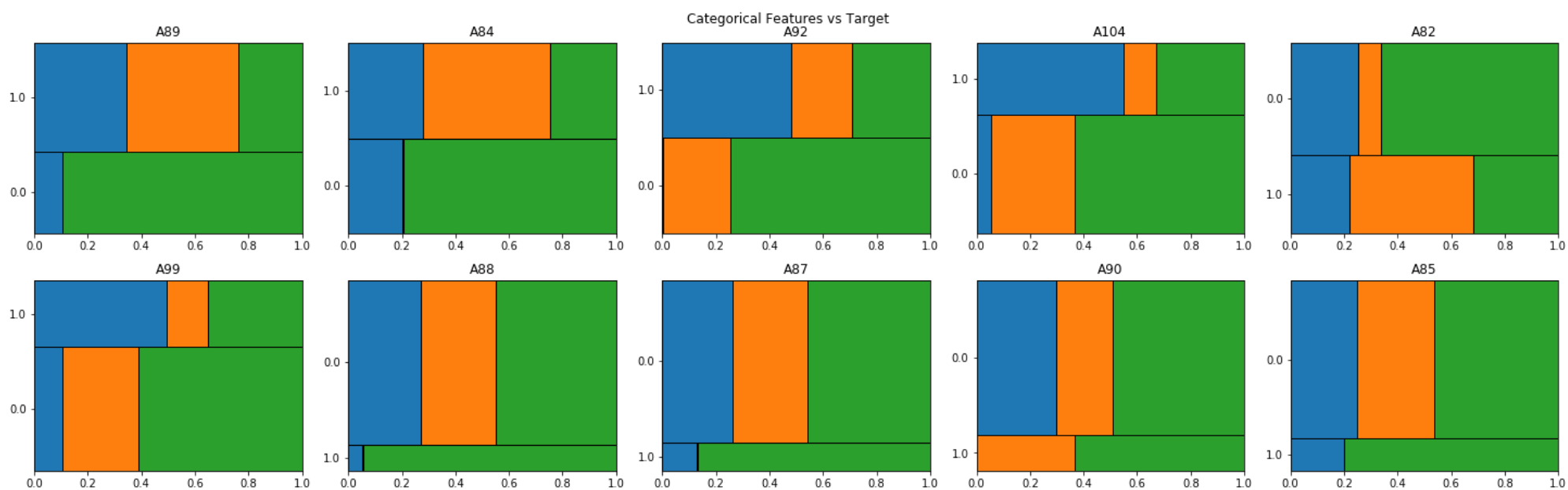


# Mosaic Plots



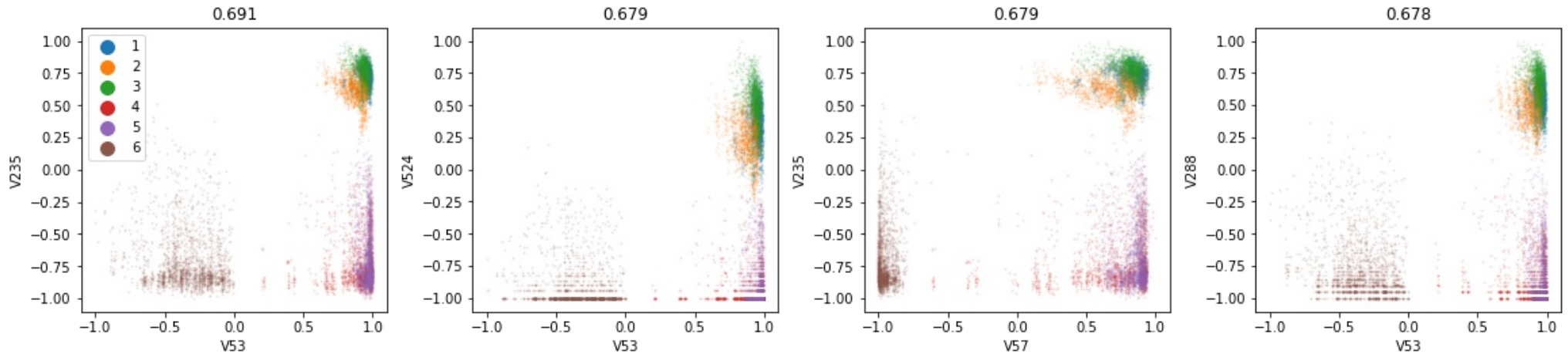


# Mosaic Plots

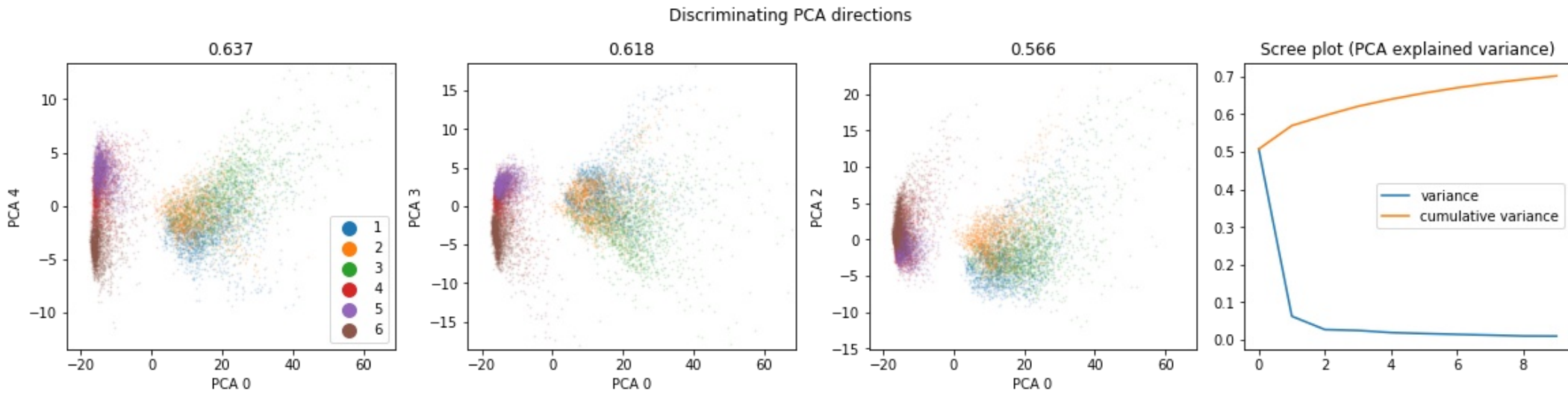


# Pairwise Plots

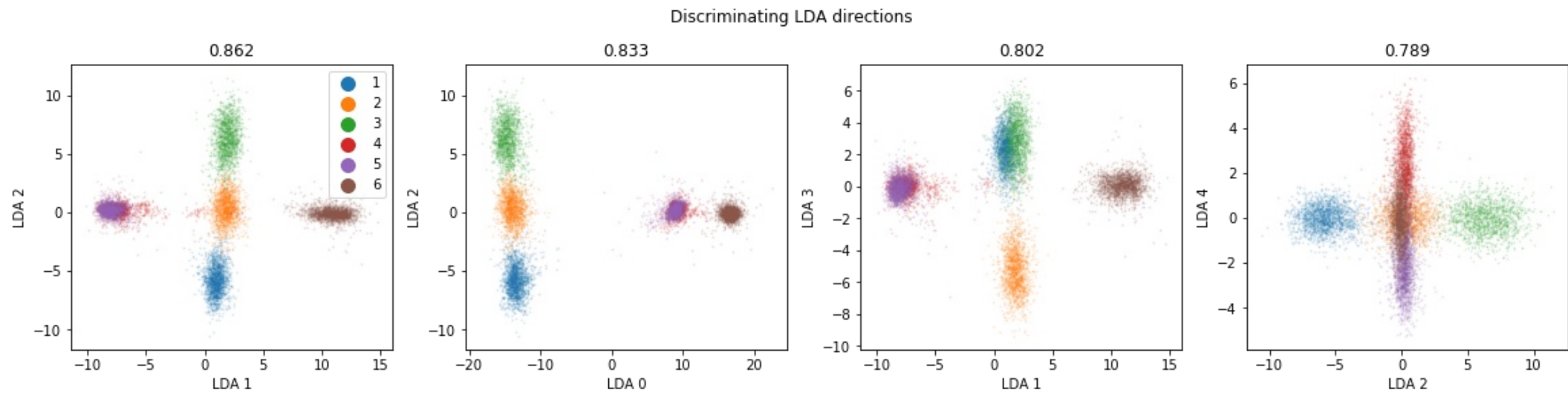
Top feature interactions



# Principal Component Analysis



# Linear Discriminant Analysis



# Preprocessing

```
X, y = ames_df.drop('SalePrice', axis=1), ames_df.SalePrice
ep = EasyPreprocessor().fit(X, y)
```

```
/home/andy/checkout/dabl/dabl/preprocessing.py:258: UserWarning: Discarding near-constant
'Land Slope', 'Condition 2', 'Roof Matl', 'Heating', 'Low Qual Fin SF', 'Kitchen AbvGr',
'rea', 'Misc Val']
    near_constant.index[near_constant.tolist()]
```

ep.ct

[illegible]

# Simple Prototypes



- Dummy Models
- Naive Bayes
- Stumps
- Linear Models



```
from dabl import SimpleClassifier
data = pd.read_csv("adult.csv", index_col=0)
SimpleClassifier().fit(data, target_col='income')
```

Either X, y  
Or dataframe, target\_col

```
/home/andy/checkout/dabl/dabl/preprocessing.py:258: UserWarning: Discarding near-constant features
near_constant.index[near_constant].tolist()))
```

```
Running DummyClassifier(strategy='prior')
```

```
accuracy: 0.759 average_precision: 0.241 f1_macro: 0.432 recall_macro: 0.500 roc_auc: 0.500
```

```
=== new best DummyClassifier(strategy='prior') (using recall_macro):
```

```
accuracy: 0.759 average_precision: 0.241 f1_macro: 0.432 recall_macro: 0.500 roc_auc: 0.500
```

```
Running GaussianNB()
```

```
accuracy: 0.407 average_precision: 0.288 f1_macro: 0.405 recall_macro: 0.605 roc_auc: 0.607
```

```
=== new best GaussianNB() (using recall_macro):
```

```
accuracy: 0.407 average_precision: 0.288 f1_macro: 0.405 recall_macro: 0.605 roc_auc: 0.607
```

```
Running MultinomialNB()
```

```
accuracy: 0.831 average_precision: 0.773 f1_macro: 0.787 recall_macro: 0.815 roc_auc: 0.908
```

```
=== new best MultinomialNB() (using recall_macro):
```

```
accuracy: 0.831 average_precision: 0.773 f1_macro: 0.787 recall_macro: 0.815 roc_auc: 0.908
```

```
Running DecisionTreeClassifier(class_weight='balanced', max_depth=1)
```

```
accuracy: 0.710 average_precision: 0.417 f1_macro: 0.682 recall_macro: 0.759 roc_auc: 0.759
```

```
Running DecisionTreeClassifier(class_weight='balanced', max_depth=5)
```

```
accuracy: 0.784 average_precision: 0.711 f1_macro: 0.750 recall_macro: 0.811 roc_auc: 0.894
```

```
Running DecisionTreeClassifier(class_weight='balanced', min_impurity_decrease=0.01)
```

```
accuracy: 0.718 average_precision: 0.561 f1_macro: 0.693 recall_macro: 0.779 roc_auc: 0.848
```

```
Running LogisticRegression(C=0.1, class_weight='balanced')
```

```
accuracy: 0.819 average_precision: 0.789 f1_macro: 0.783 recall_macro: 0.832 roc_auc: 0.915
```

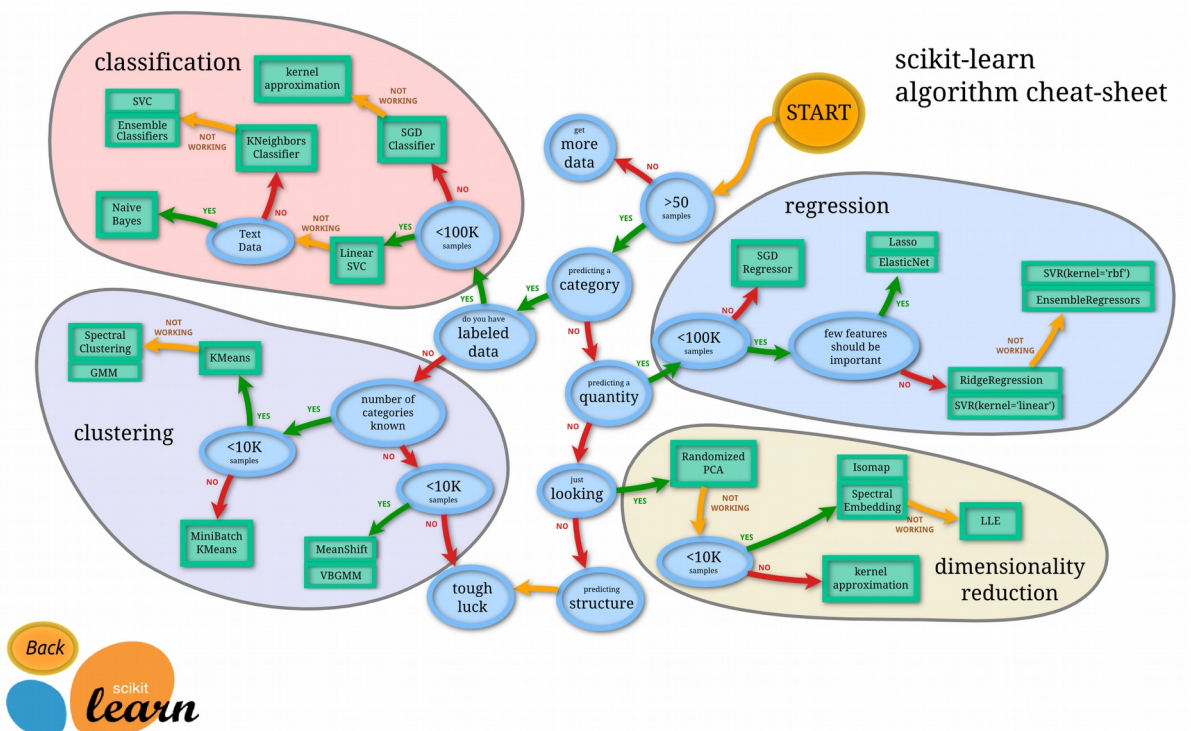
```
=== new best LogisticRegression(C=0.1, class_weight='balanced') (using recall_macro):
```

```
accuracy: 0.819 average_precision: 0.789 f1_macro: 0.783 recall_macro: 0.832 roc_auc: 0.915
```

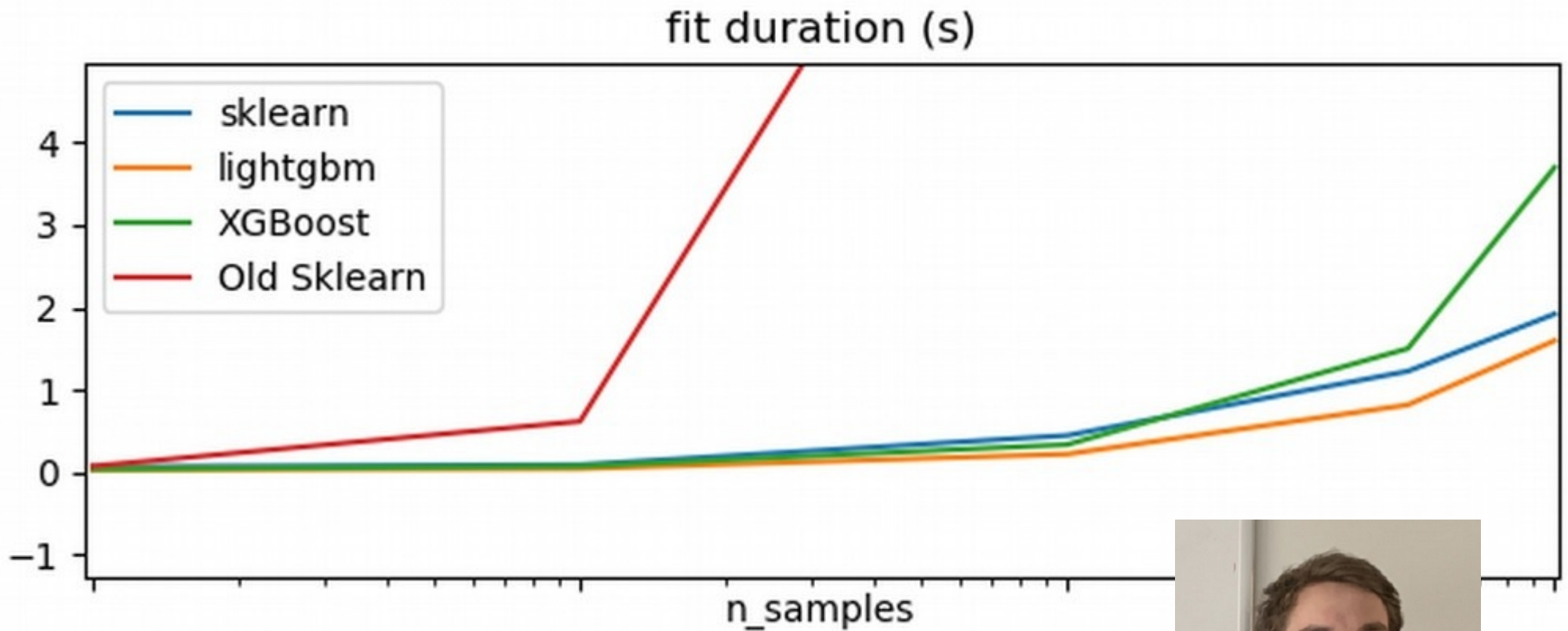
# Automatic Model Search

# Complex Models

- More Linear Models
- Random forest
- Gradient boosting
- Kernel methods



# Side note: HistGradientBoosting



# Successive Halving

- Given  $n$  configuration and budget  $B$
- pick  $\eta=2$  or  $\eta=3$  (wording follows 2)
- Each iteration, keep best half of configurations

## SUCCESSIVEHALVING (Finite horizon)

**input:** Budget  $B$ , and  $n$  arms where  $\ell_{i,k}$  denotes the  $k$ th loss from the  $i$ th arm, maximum size  $R$ ,  $\eta \geq 2$  ( $\eta = 3$  by default).

**Initialize:**  $S_0 = [n]$ ,  $s = \min\{t \in \mathbb{N} : nR(t+1)\eta^{-t} \leq B, t \leq \log_\eta(\min\{R, n\})\}$ .

**For**  $k = 0, 1, \dots, s$

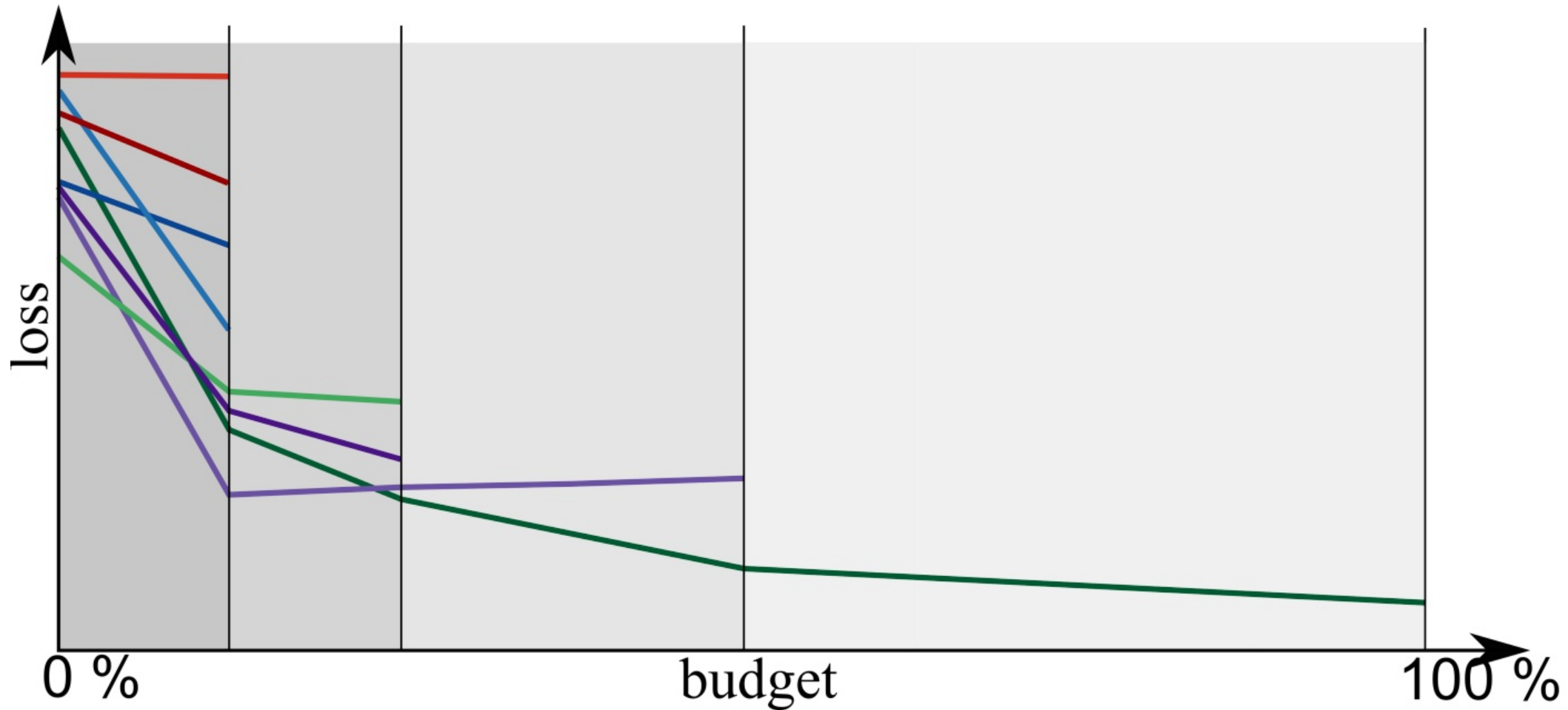
Set  $n_k = \lfloor n\eta^{-k} \rfloor$ ,  $r_k = \lfloor R\eta^{k-s} \rfloor$

Pull each arm in  $S_k$  for  $r_k$  times.

Keep the best  $\lfloor n\eta^{-(k+1)} \rfloor$  arms in terms of the  $r_k$ th observed loss as  $S_{k+1}$ .

**Output :**  $\hat{i}, \ell_{\hat{i},R}$  where  $\hat{i} = \arg \min_{i \in S_{s+1}} \ell_{i,R}$

# Successive Halving Illustrated



# Portfolio creation

- Run Hyper-parameter optimization across models on large benchmark suite (OpenML-CC18)
- Evaluate all final models across all datasets
- Greedily create portfolio of best-performing, diverse models
- “Practical Automated Machine Learning for the AutoML Challenge 2018” Feurer et. al.



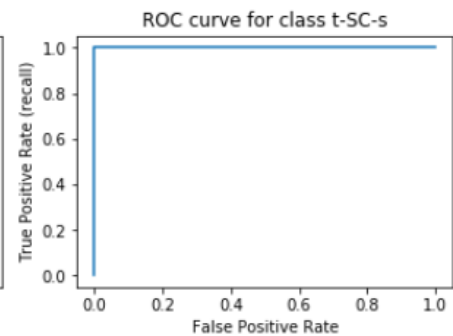
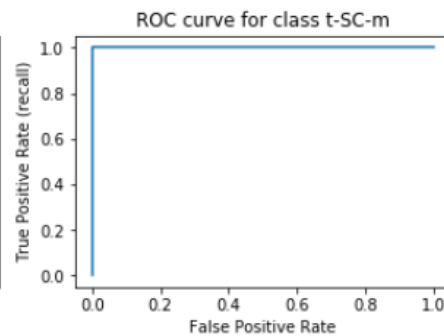
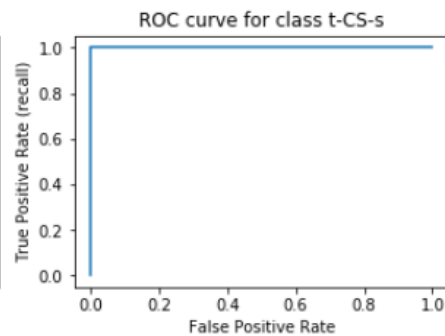
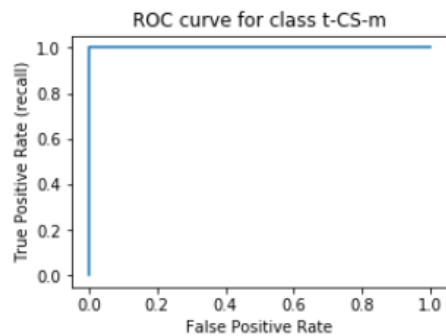
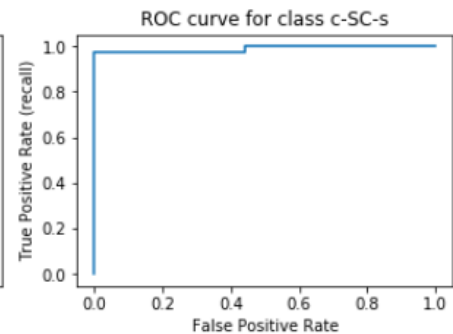
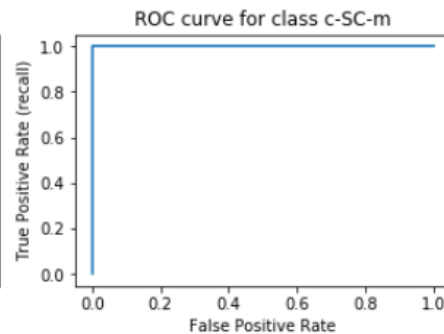
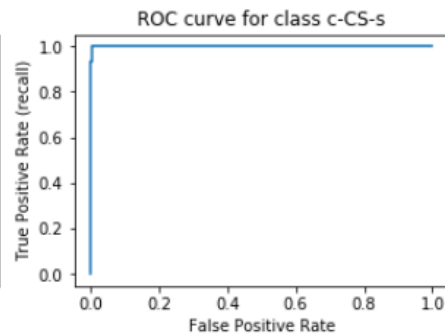
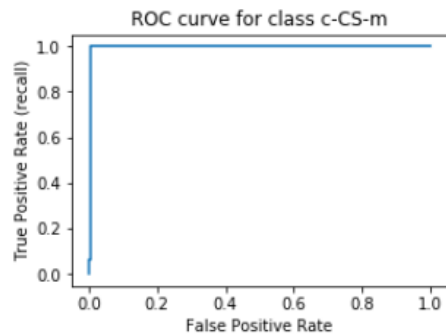
# Model Explanation

```
from sklearn.model_selection import train_test_split
df_train, df_test = train_test_split(data)
ac = AnyClassifier().fit(df_train, target_col='target')
```

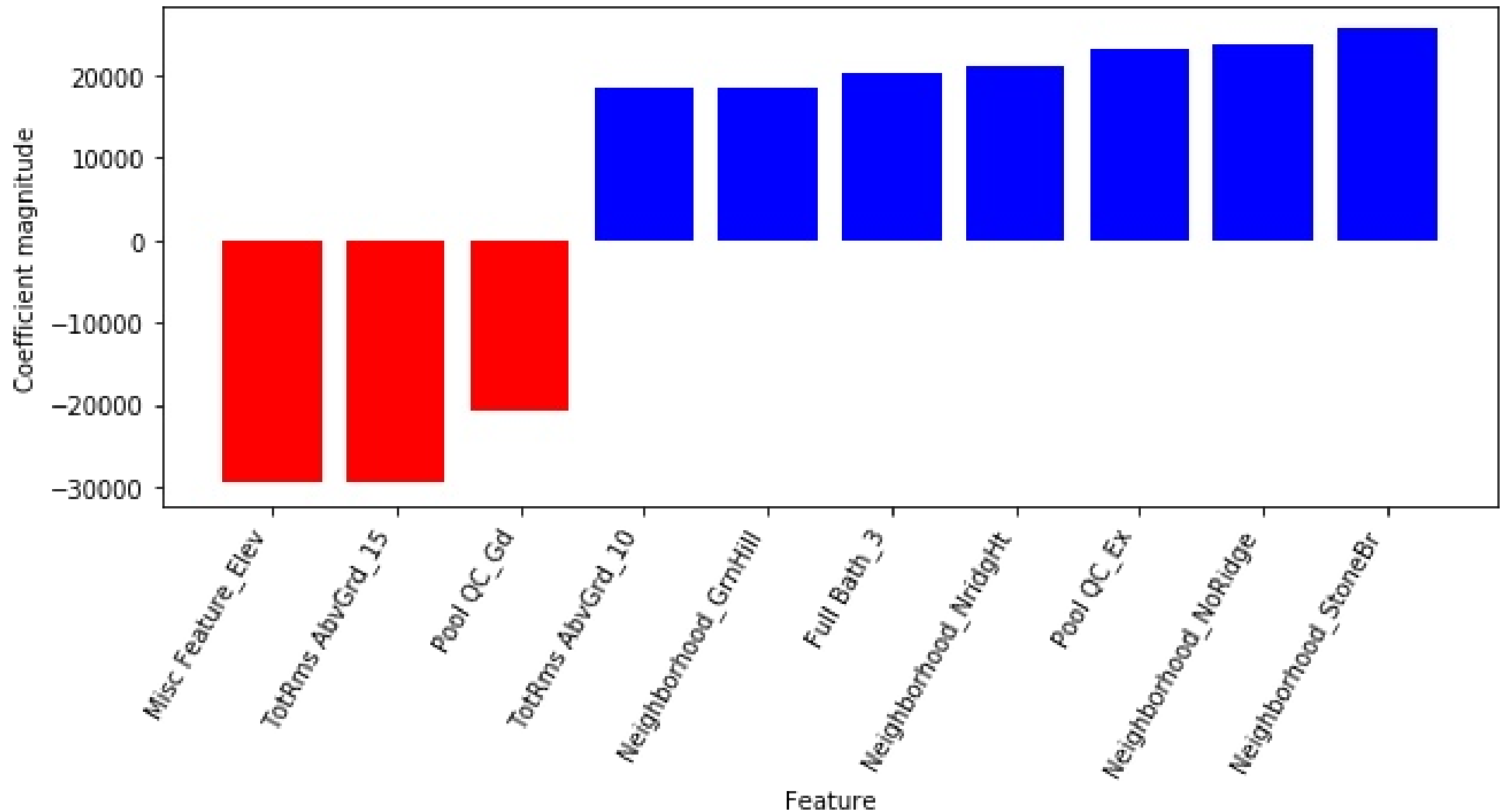
```
import dabl
dabl.explain(ac, X_val=df_test, target_col='target')
```

# Metrics

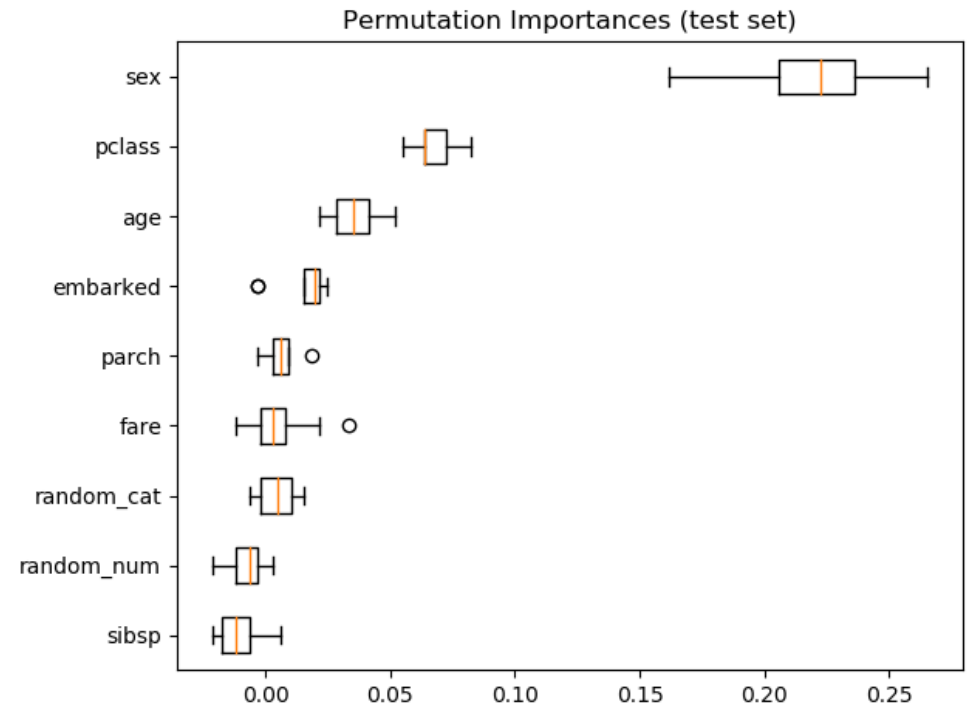
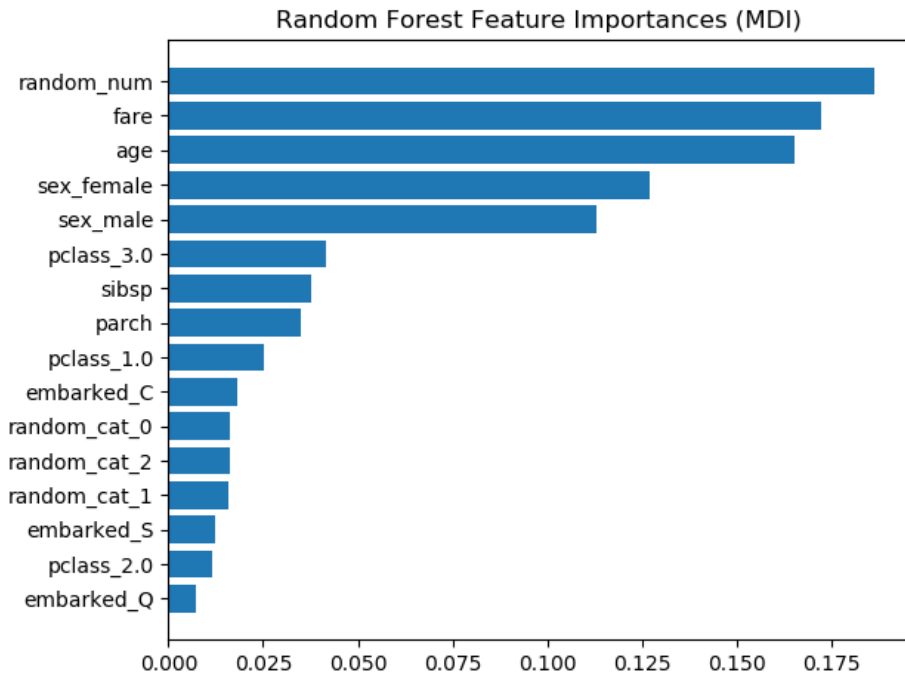
	precision	recall	f1-score	support	
c-CS-m	0.97	0.97	0.97	32	[[31 1 0 0 0 0 0 0] [ 0 31 0 0 0 0 0 0] [ 0 0 37 0 0 0 0 0] [ 1 0 0 32 0 0 0 1] [ 0 1 0 0 40 0 0 0] [ 0 0 0 0 0 21 0 0] [ 0 0 0 0 0 0 33 0] [ 0 0 0 0 0 0 0 41]]
c-CS-s	0.94	1.00	0.97	31	
c-SC-m	1.00	1.00	1.00	37	
c-SC-s	1.00	0.94	0.97	34	
t-CS-m	1.00	0.98	0.99	41	
t-CS-s	1.00	1.00	1.00	21	
t-SC-m	1.00	1.00	1.00	33	
t-SC-s	0.98	1.00	0.99	41	
accuracy			0.99	270	
macro avg	0.99	0.99	0.99	270	
weighted avg	0.99	0.99	0.99	270	



# Coefficients / Feature importances

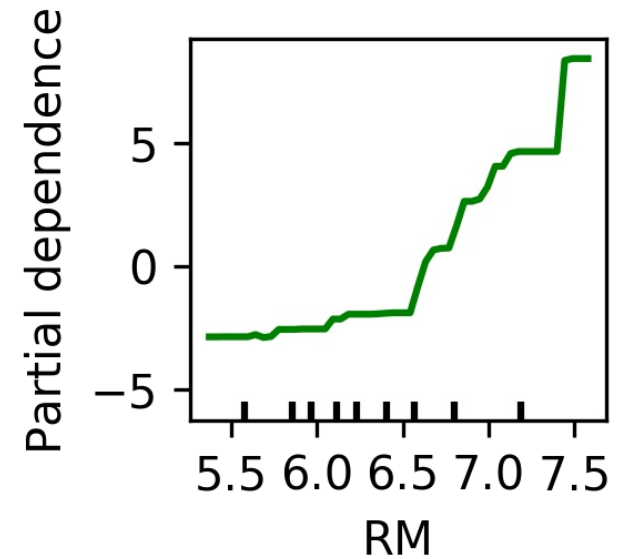
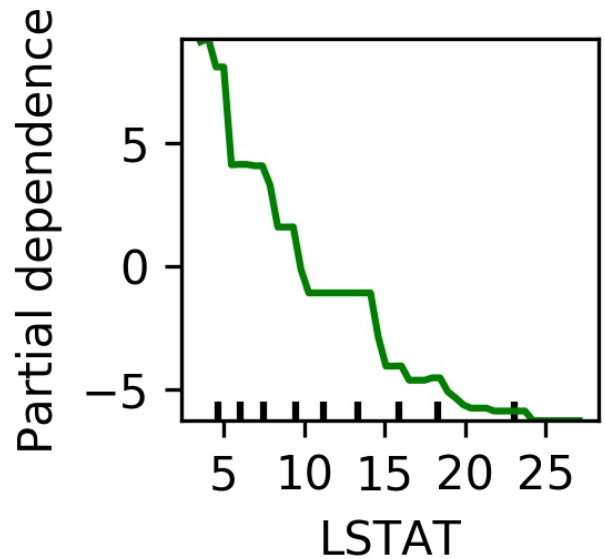
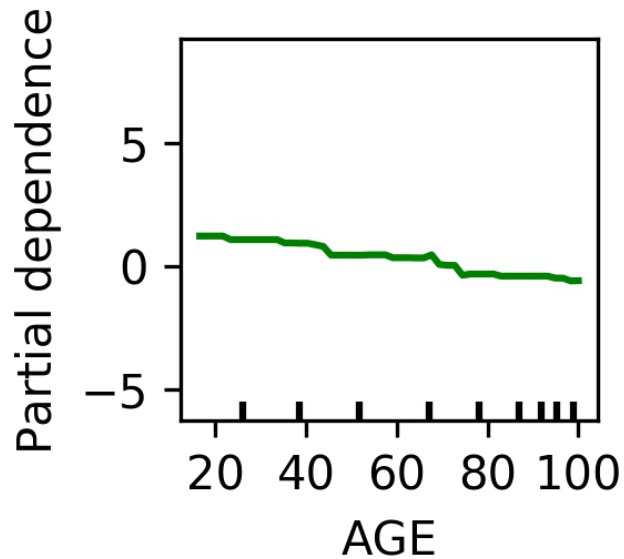
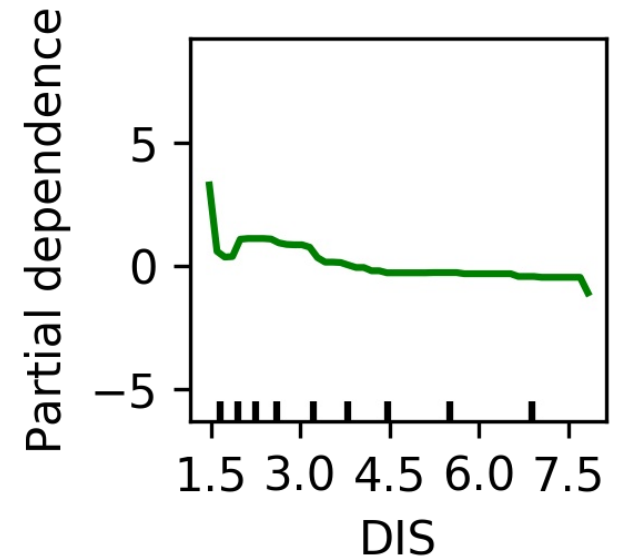
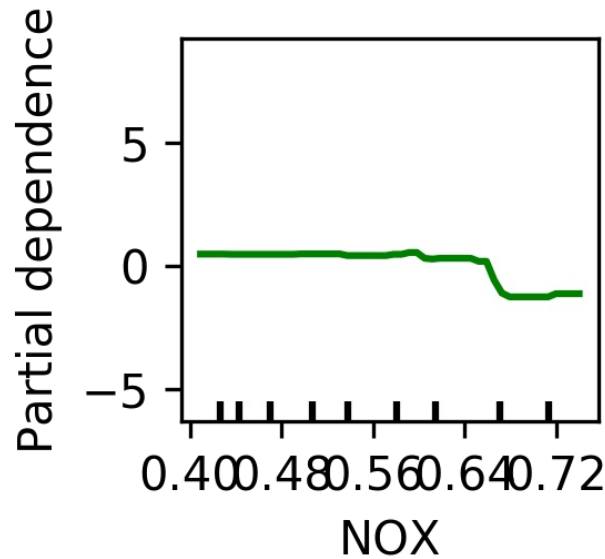
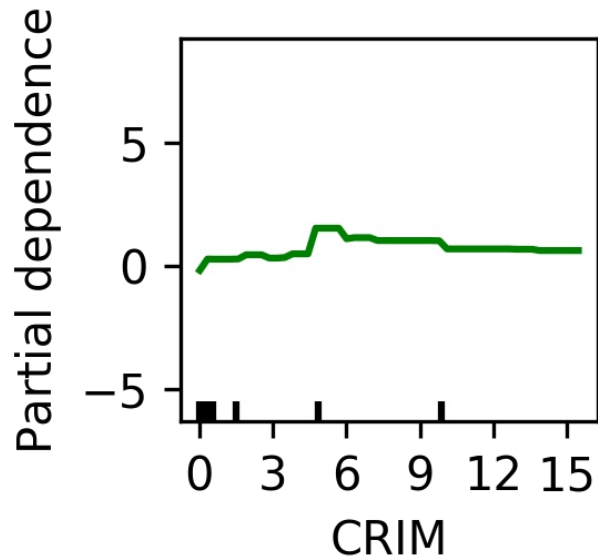


# Permutation Importance

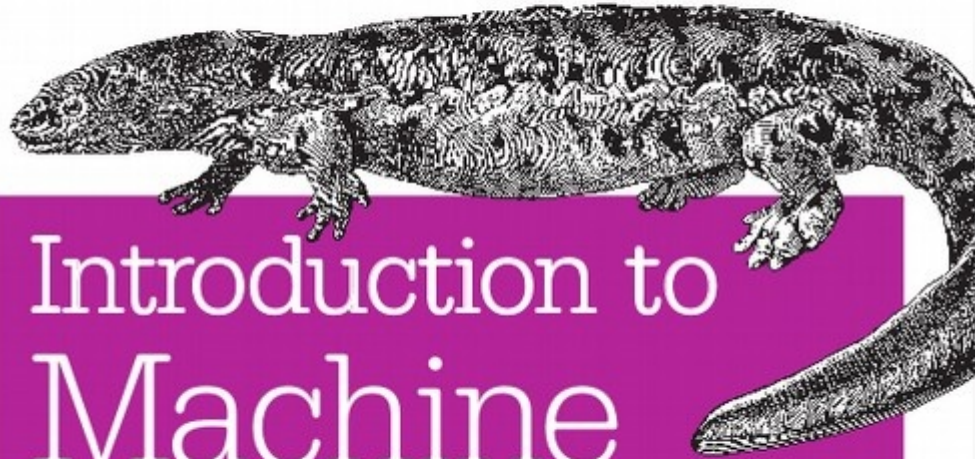


[https://scikit-learn.org/dev/auto\\_examples/inspection/plot\\_permutation\\_importance.html](https://scikit-learn.org/dev/auto_examples/inspection/plot_permutation_importance.html)

# Partial Dependence Plots



O'REILLY®



# Introduction to Machine Learning with Python

A GUIDE FOR DATA SCIENTISTS

Andreas C. Müller & Sarah Guido



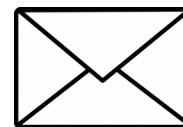
[amueller.github.io](https://amueller.github.io)



[@amuellerm1](https://twitter.com/amuellerm1)



[@amueller](https://github.com/amueller)



[andreas.mueller  
@columbia.com](mailto:andreas.mueller@columbia.com)